编译原理

实验二：自底向上的语法分析LR(1)

规格严格，功夫到家

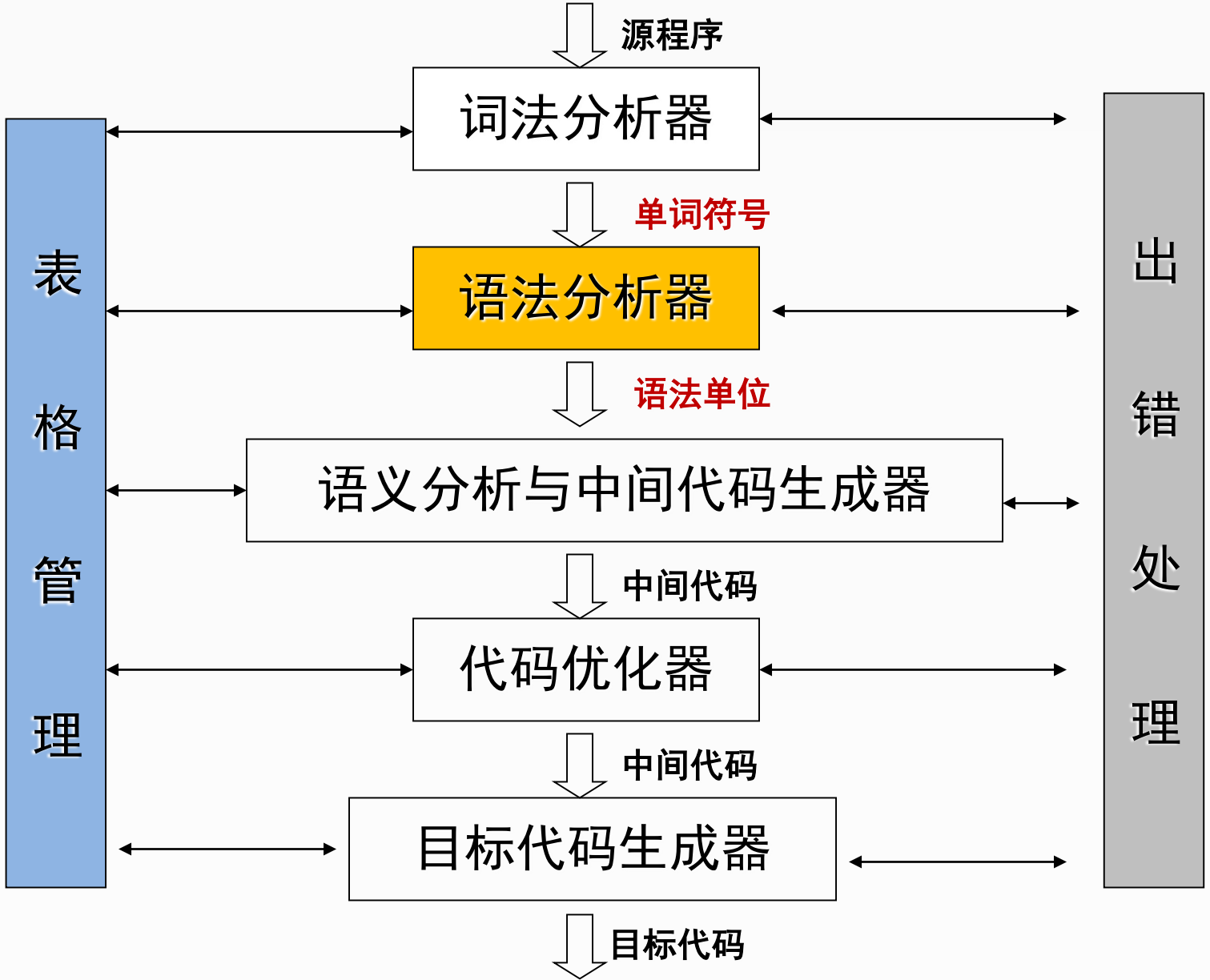
编译程序的总体结构

✔实验一：

✔实验二：

✔实验三：

✔实验四：



实验目的

1. 深入了解语法分析程序实现原理及方法。

2. 理解LR(1)分析法是严格的从左向右扫描和自底向上的语法分析方法。

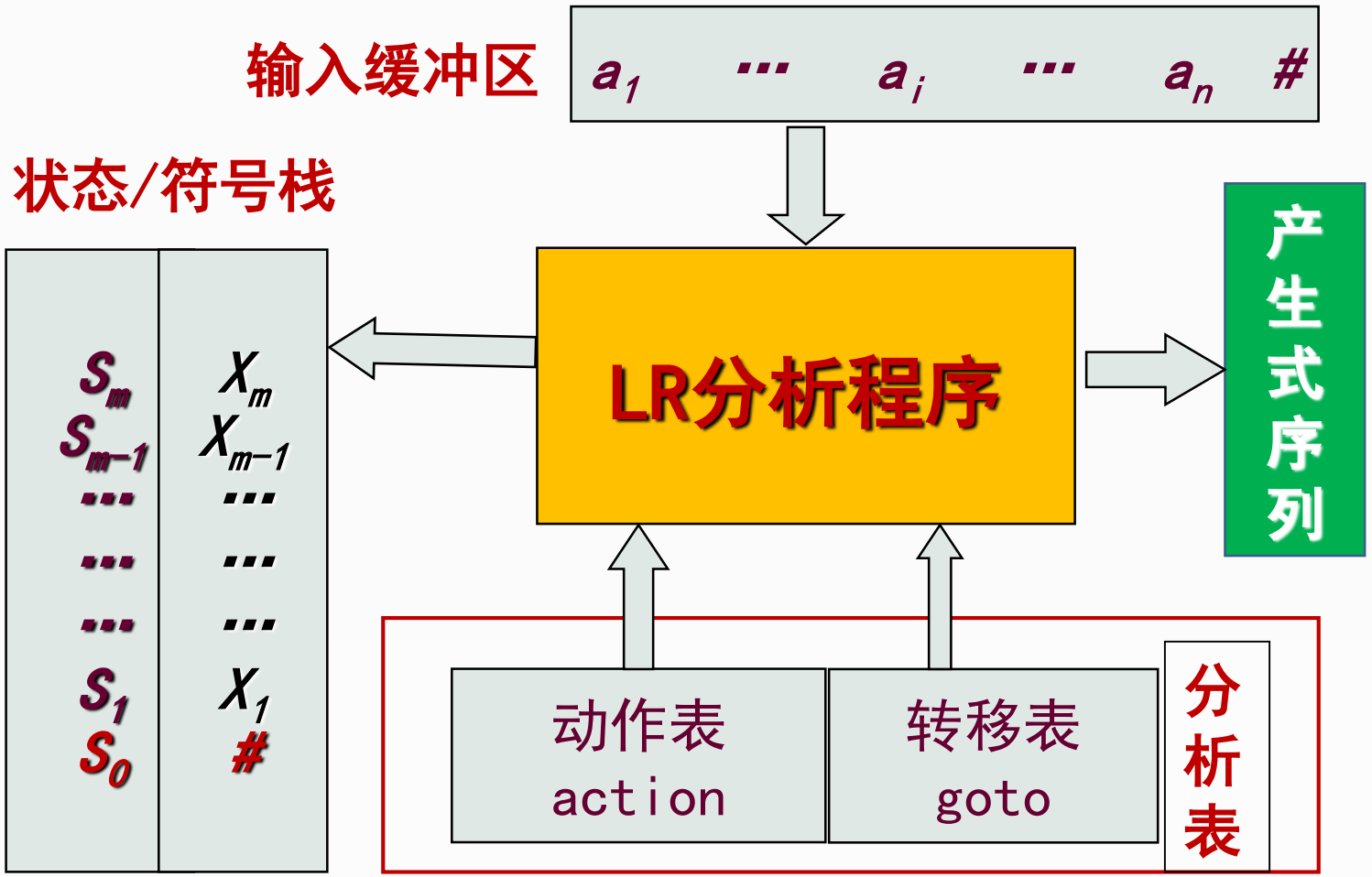
实验学时数：8学时。

实验内容

1. 利用LR(1)分析法，设计语法分析程序，对实验一输出的单词符号串进行语法分析；
2. 输出推导过程中所用产生式序列并保存在输出文件中；
3. 基本要求：实验模板代码中支持变量声明、变量赋值、基本算术运算的文法；
4. 进阶要求：自行设计更多文法并完成实验，需体现在实验报告中。

说明： 实验一测试用例较复杂的情况下，实验二也需要同学自行定义文法并生成LR（1）分析表来完成实验。

LR语法分析器的总体结构



两个输入

- ✓ Token串
- ✓ LR分析表

两个栈

- ✓ 状态栈
- ✓ 符号栈

四种动作

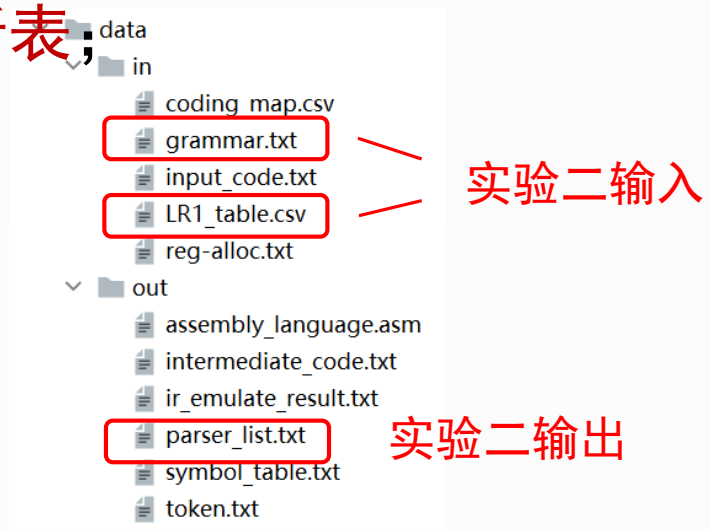
- ✓ **移入：**
  将下一输入符号移入栈
- ✓ **归约：**
  用产生式左侧的非终结符替换栈顶的句柄（某产生式右部）
- ✓ **接受：**分析成功
- ✓ **出错：**出错处理

一个输出

- ✓ **产生式序列**

实验总体步骤

1. 定义描述程序设计语言语法的文法，并编写拓广文法；
2. 构造LR（1）分析表（借助编译工作台完成）；
3. 读入文法、LR（1）分析表；
4. 编写LR（1）驱动程序完成LR分析器移进、归约、出错、接受四个动作；
5. 输入：实验一输出的单词符号串、文法和LR1分析表；
6. 输出：产生式序列并保存在文件中。



1

定义文法

定义描述程序设计语言语法的文法，并编写拓广文法

P -> S_list;

S_list -> S Semicolon S_list;

S_list -> S Semicolon;

S -> D id;

D -> int;

S -> id = E;

S -> return E;

E -> E + A;

E -> E - A;

E -> A;

A -> A * B;

A -> B;

B -> (E);

B -> id;

B -> IntConst;

- 文法保存位置：data/in/grammar.txt

2 利用编译工作台构造LR（1）分析表

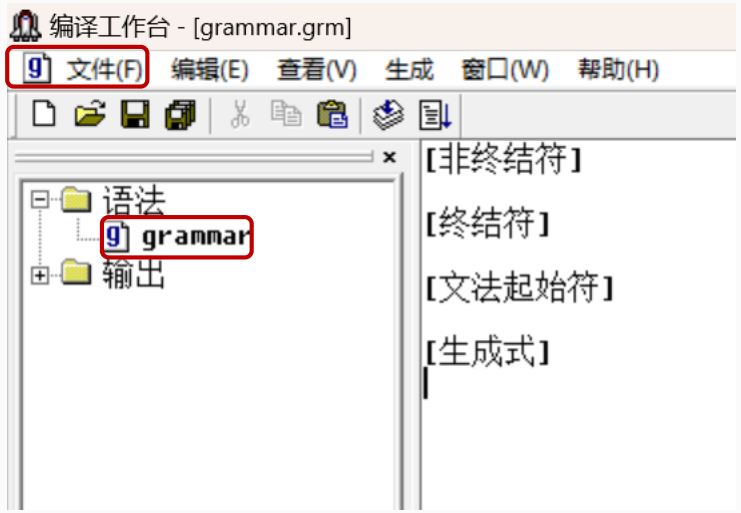
生成分析表步骤：

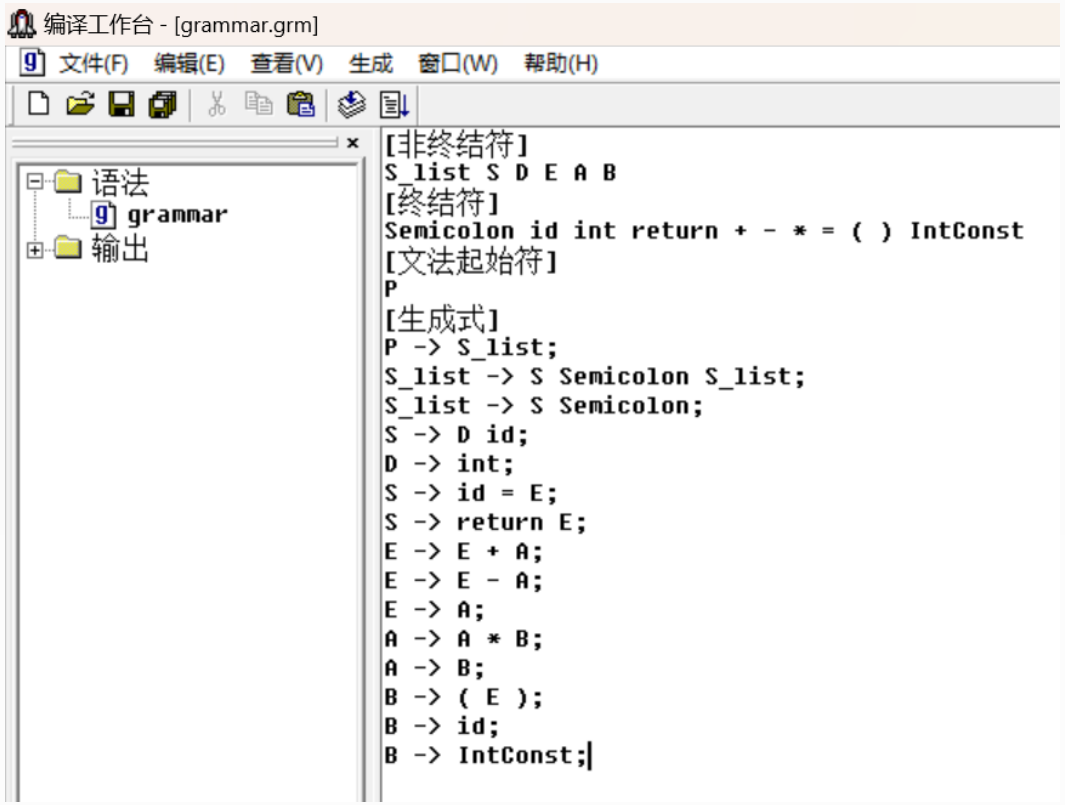
1. Windows环境安装编译工作台，下载链接：

https://gitee.com/hitsz-cslab/Compiler/releases/tag/2022F.0.1

2 利用编译工作台构造LR（1）分析表

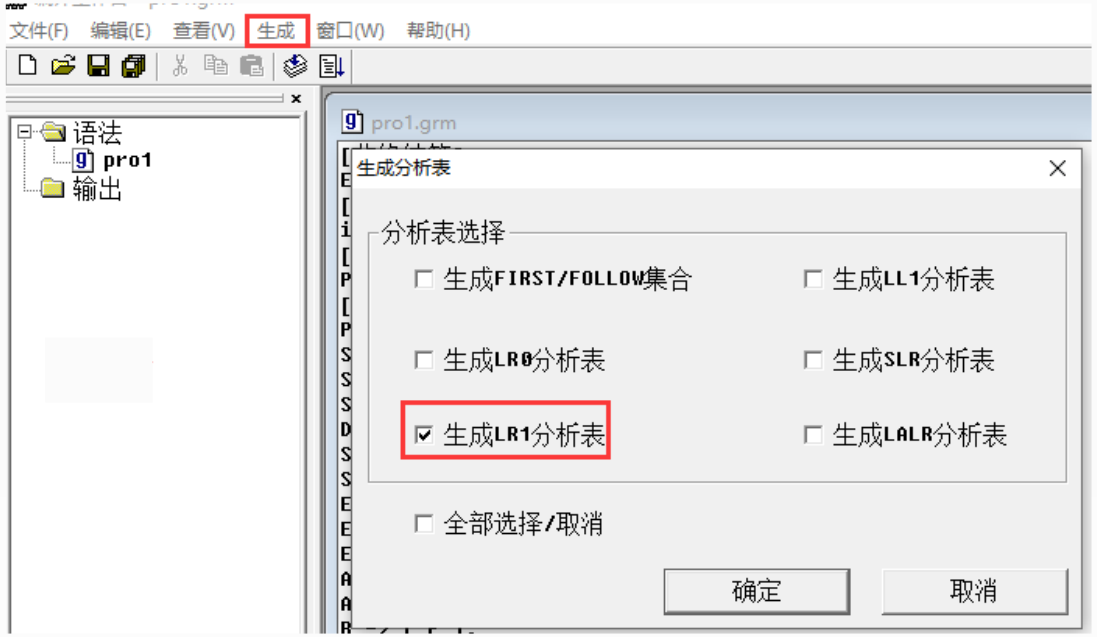
2. 新建工程，创建一个语法文件，按照**文件模板**填写；

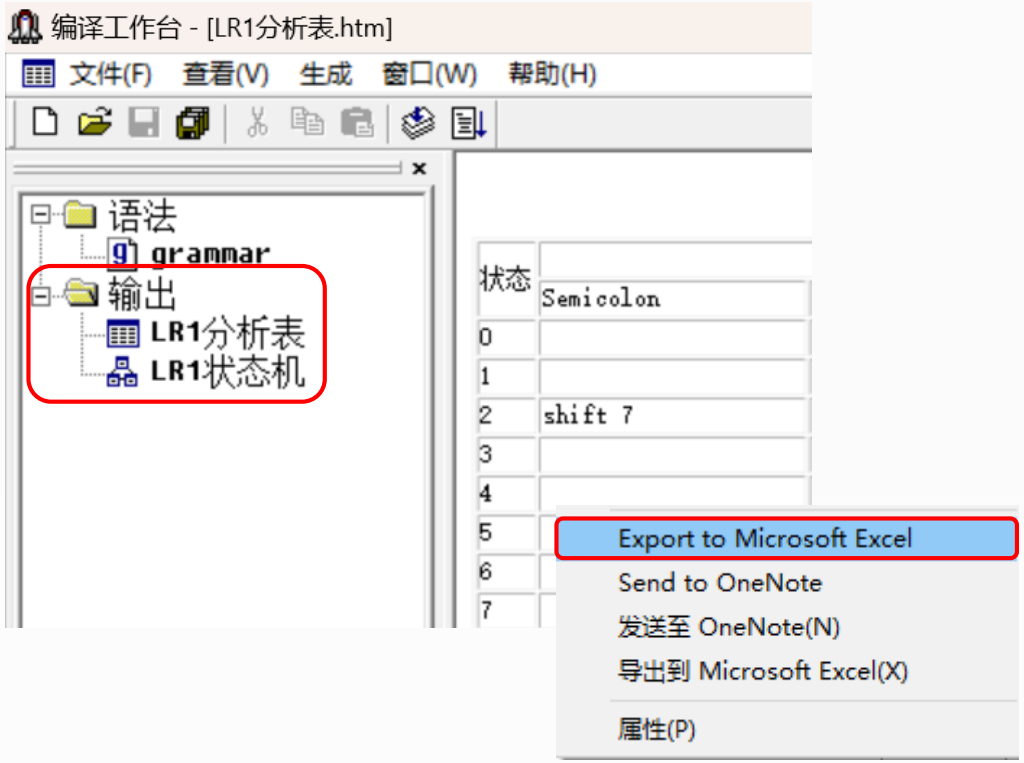




2 利用编译工作台构造LR（1）分析表

3. 点击菜单“生成”——“生成分析表”；在分析表上右击可以导出分析表；





参考文法的LR(1)分析表——包括ACTION和GOTO

	LR1分析表																	
	状态	ACTION												GOTO					
		id	(	)	+	-	*	=	int	return	IntConst	Semicolon	$	E	S_list	S	A	B	D
5	0	shift 4							shift 5	shift 6					1	2			3
6	1												accept						
7	2											shift 7							
8	3	shift 8																	
9	4							shift 9											
10	5	reduce D -> int																	
11	6	shift 13	shift 14								shift 15			10			11	12	
12	7	shift 4							shift 5	shift 6			reduce S_list ->	16	2				3
13	8											reduce S -> D id							
14	9	shift 13	shift 14								shift 15			17			11	12	
15	10				shift 18	shift 19						reduce S -> return E							
16	11				reduce E ->	reduce E ->	shift 20					reduce E -> A							
17	12				reduce A ->	reduce A ->	reduce A -> B					reduce A -> B							
18	13				reduce B ->	reduce B ->	reduce B -> id					reduce B -> id							
19	14	shift 24	shift 25								shift 26			21			22	23	
20	15				reduce B ->	reduce B ->	reduce B -> IntConst					reduce B -> IntConst							
21	16												reduce S_list -> S Semicolon S_list						
22	17				shift 18	shift 19						reduce S -> id = E							
23	18	shift 13	shift 14								shift 15						27	12	
24	19	shift 13	shift 14								shift 15						28	12	
25	20	shift 13	shift 14								shift 15							29	
26	21			shift 30	shift 31	shift 32													
27	22			reduce E ->	reduce E ->	reduce E ->	shift 33												
28	23			reduce A ->	reduce A ->	reduce A ->	reduce A -> B												
29	24			reduce B ->	reduce B ->	reduce B ->	reduce B -> id												
30	25	shift 24	shift 25								shift 26			34			22	23	
31	26			reduce B ->	reduce B ->	reduce B ->	reduce B -> IntConst												
32	27				reduce E ->	reduce E ->	shift 20					reduce E -> E + A							
33	28				reduce E ->	reduce E ->	shift 20					reduce E -> E - A							
34	29				reduce A ->	reduce A ->	reduce A -> A * B					reduce A -> A * B							
35	30				reduce B ->	reduce B ->	reduce B -> (E)					reduce B -> (E)							
36	31	shift 24	shift 25								shift 26						35	23	
37	32	shift 24	shift 25								shift 26						36	23	
38	33	shift 24	shift 25								shift 26							37	
39	34			shift 38	shift 31	shift 32													
40	35			reduce E ->	reduce E ->	reduce E ->	shift 33												
41	36			reduce E ->	reduce E ->	reduce E ->	shift 33												
42	37			reduce A ->	reduce A ->	reduce A ->	reduce A -> A * B												
43	38			reduce B ->	reduce B ->	reduce B ->	reduce B -> (E)												

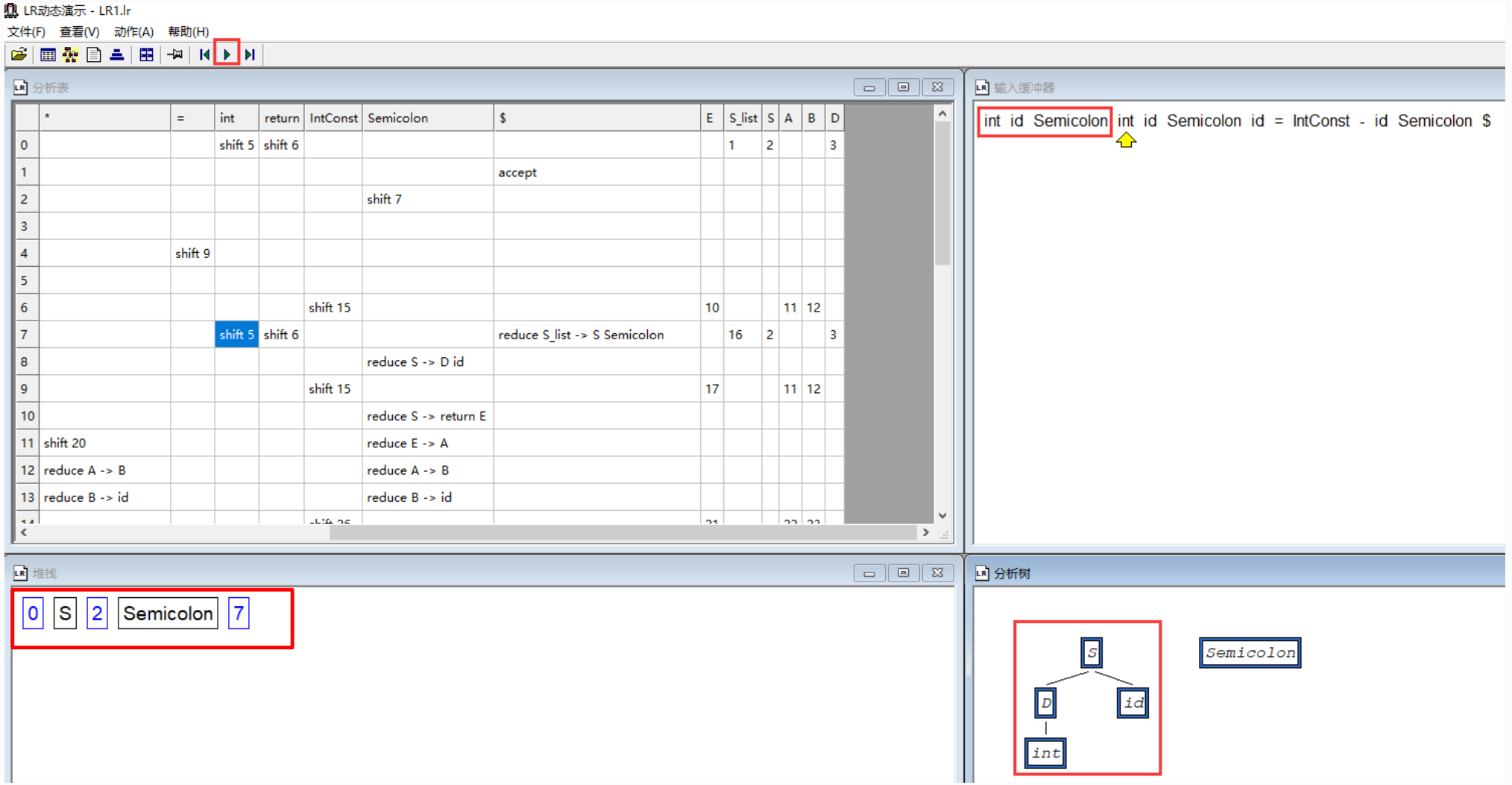
借助LR分析表动态分析源文件

举例：int a; int c; c = 3-a;

1. 新建一个源文件，输入要分析的源文件（将源代码替换成编码表内的种别码）；

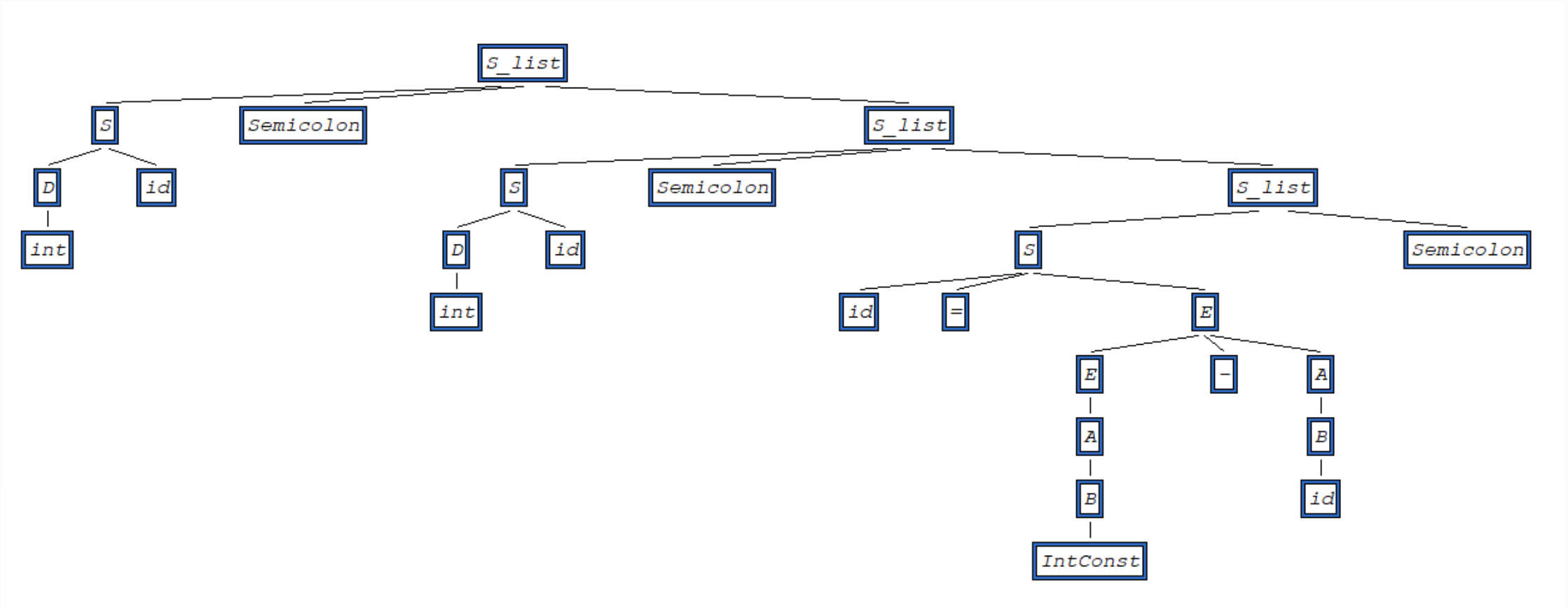
2. 点击生成——动态分析；

编译工作台动态分析源代码



	*	=	int	return	IntConst	Semicolon	$	E	S_list	S	A	B	D
0			shift 5	shift 6					1	2			3
1							accept						
2						shift 7							
3													
4		shift 9											
5													
6					shift 15			10			11	12	
7			shift 5	shift 6			reduce S_list -> S Semicolon		16	2			3
8						reduce S -> D id							
9					shift 15			17			11	12	
10						reduce S -> return E							
11	shift 20					reduce E -> A							
12	reduce A -> B					reduce A -> B							
13	reduce B -> id					reduce B -> id							

动态分析结束生成完整的语法树



语法分析程序输出结果说明

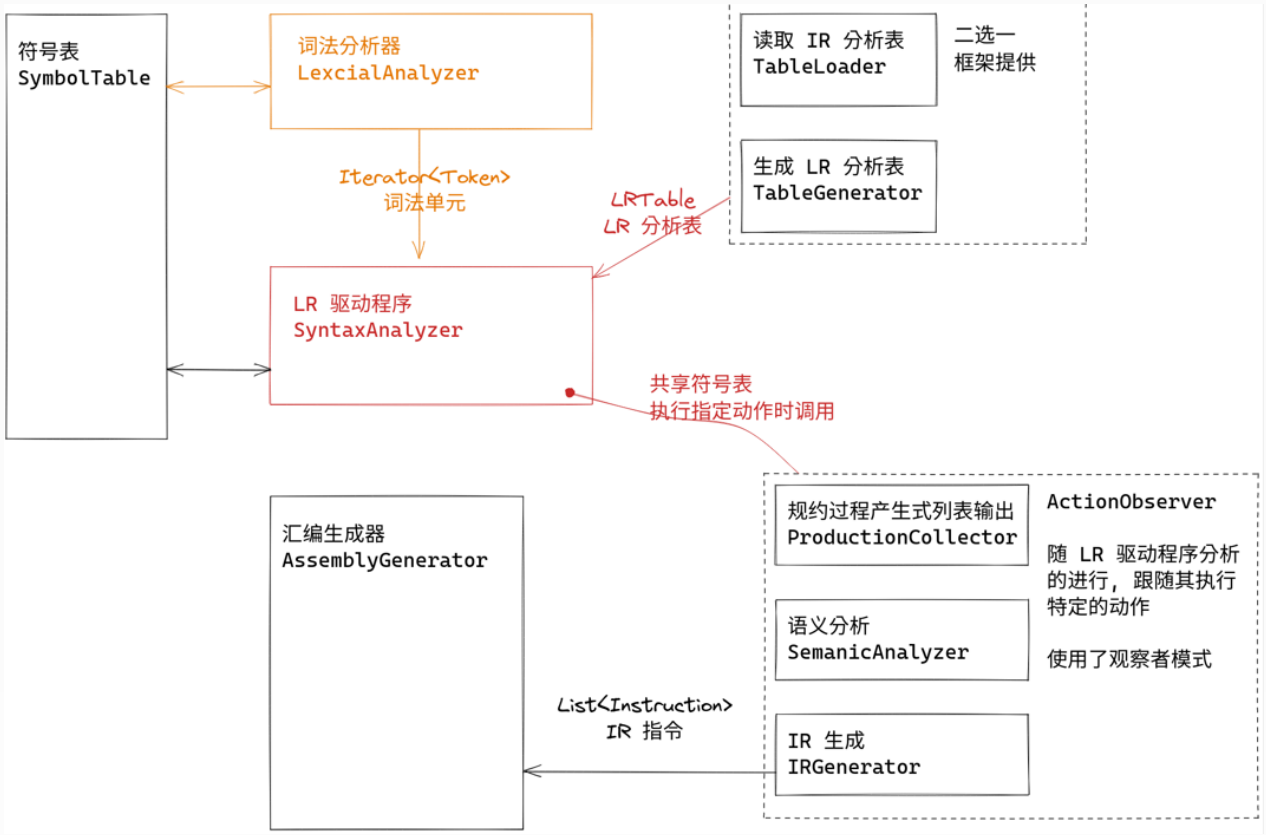
在每一次归约时，顺序输出归约所用到的产生式，故语法分析的输出是产生式序列。

举例：int a; int c; c = 3-a;

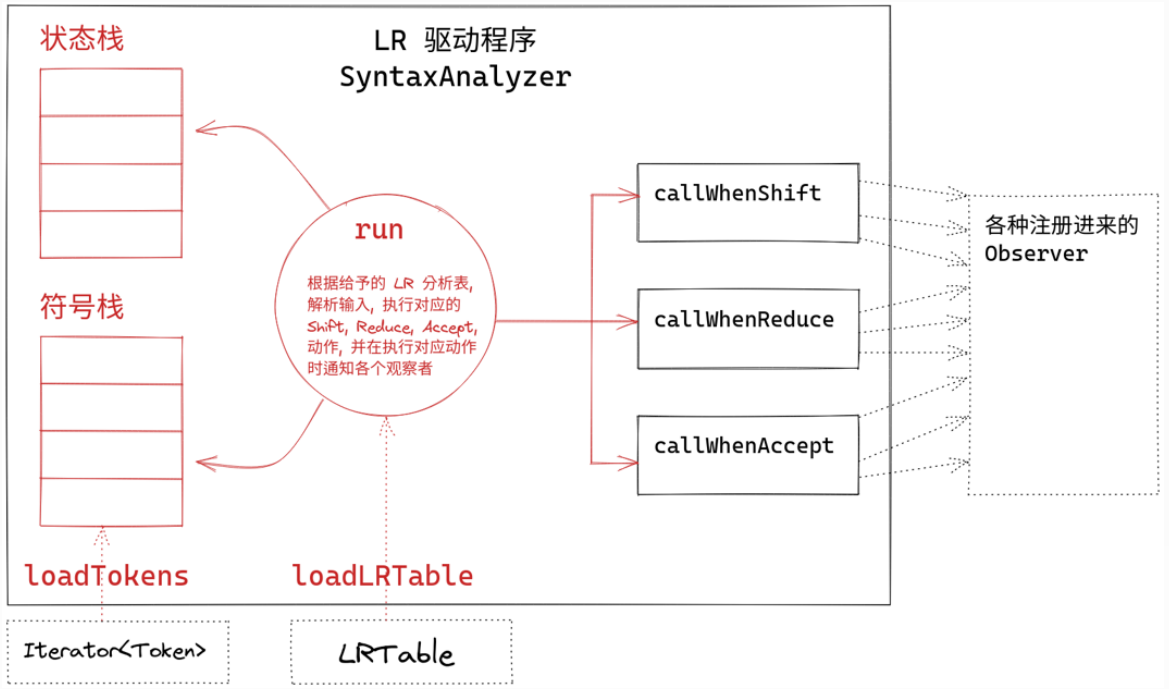
语法分析输出产生式列表 →

D -> int
S -> D id
D -> int
S -> D id
B -> IntConst
A -> B
E -> A
B -> id
A -> B
E -> E - A
S -> id = E
S_list -> S Semicolon
S_list -> S Semicolon S_list
S_list -> S Semicolon S_list
P -> S_list

代码框架

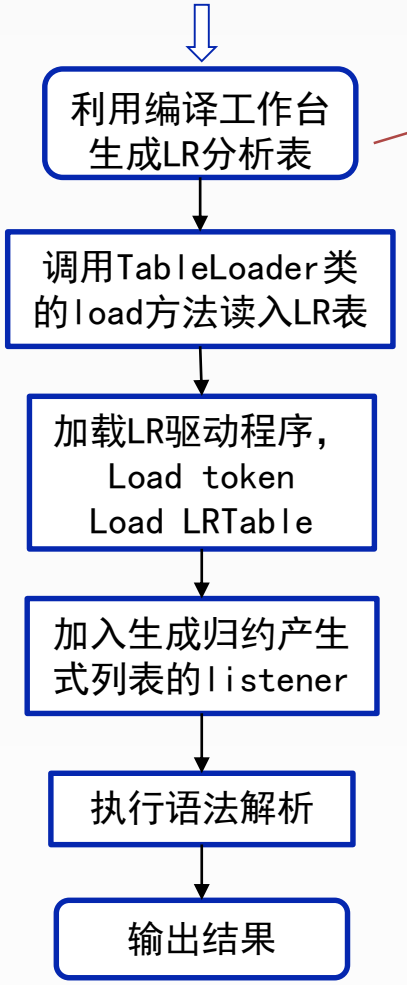


编译器程序整体框架图



LR 驱动程序框架图

主程序流程说明



用自己生成的LR分析表替换模板中的LR1_table.csv，如果不改变拓广文法，可以不用替换。请注意表格格式：CSV

```
final var tableLoader = new TableLoader();
final var lrTable = tableLoader.load(FilePathConfig.LR1_TABLE_PATH);
```

模板已完成

```
final var parser = new SyntaxAnalyzer(symbolTable);
parser.loadTokens(tokens); //TODO
parser.loadLRTable(lrTable); //TODO
```

```
final var productionCollector = new
ProductionCollector(GrammarInfo.getBeginProduction());
parser.registerObserver(productionCollector);
```

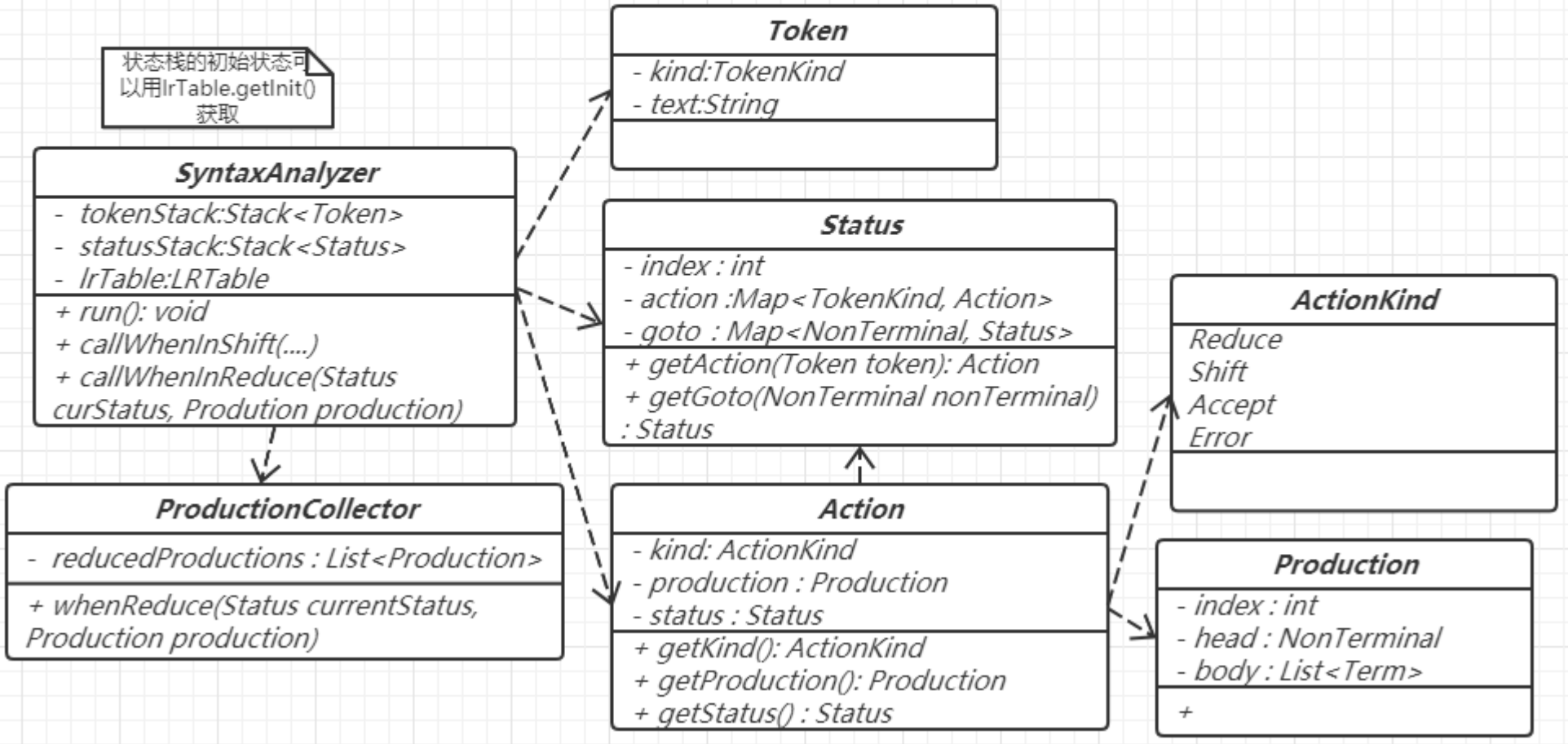
GrammarInfo里完成了产生式加载，产生式数据结构类：Production
模板已完成

```
parser.run(); //TODO
```

```
productionCollector.dumpToFile(FilePathConfig.PARSER_PATH);
```

模板已完成

语法分析过程描述：
1. 建立符号栈和状态栈，初始化栈；
2. 根据状态栈栈顶元素和待读入的下一个token查询判断下一个待执行动作；
3. 如果是Shift，把Action的状态压入状态栈，对应的token压入符号栈；
4. 如果是Reduce，根据产生式长度，符号栈和状态栈均弹出对应长度个token和状态；产生式左侧的非终结符压入符号栈，根据符号栈和状态栈栈顶状态获取Goto表的状态，压入状态栈；
5. 如果是Accept，语法分析执行结束；
6. ProductionCollector观察者内部顺序记录归约所用到的产生式，语法分析结束输出到文件。



同学们，
请开始实验